CPE
Arquivos

Departamento de Engenharia Elétrica – UnB

Roteiro

Introdução

Até agora...

Nossos programas lêem dados do teclado e imprimem resultados na tela do computador.

Objetivo desta aula

- Melhorar nossos programas para também ler dados de arquivos e gravar os resultados em arquivos.
  - O programa não precisará depender só de dados informados diretamente pelo usuário no teclado, em toda execução.
  - Armazenamento de resultados do programa no disco rígido.
  - Recuperação de informações já processadas.

Arquivos

Características

- Podem armazenar grande quantidade de informação.
- Dados são persistentes (gravados em disco).
- Acessso aos dados pode ser não seqüencial (acesso direto a registros em um banco de dados).
- Acesso à informação pode ser concorrente (mais de um programa pode acessar ao mesmo tempo).

Nomes e extensões

- Arquivos são identificados por um nome.
- O nome de um arquivo pode conter uma extensão que indica o conteúdo do arquivo.

Algumas extensões

arq.txt	arquivo texto simples
arq.cpp	código fonte em C++
arq.pdf	*portable document format*
arq.html	arquivo para páginas WWW *(hypertext markup language)*
arq*	arquivo executável (UNIX, Linux, Mac)
arq.exe	arquivo executável (Windows)

Tipos de arquivos

Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:

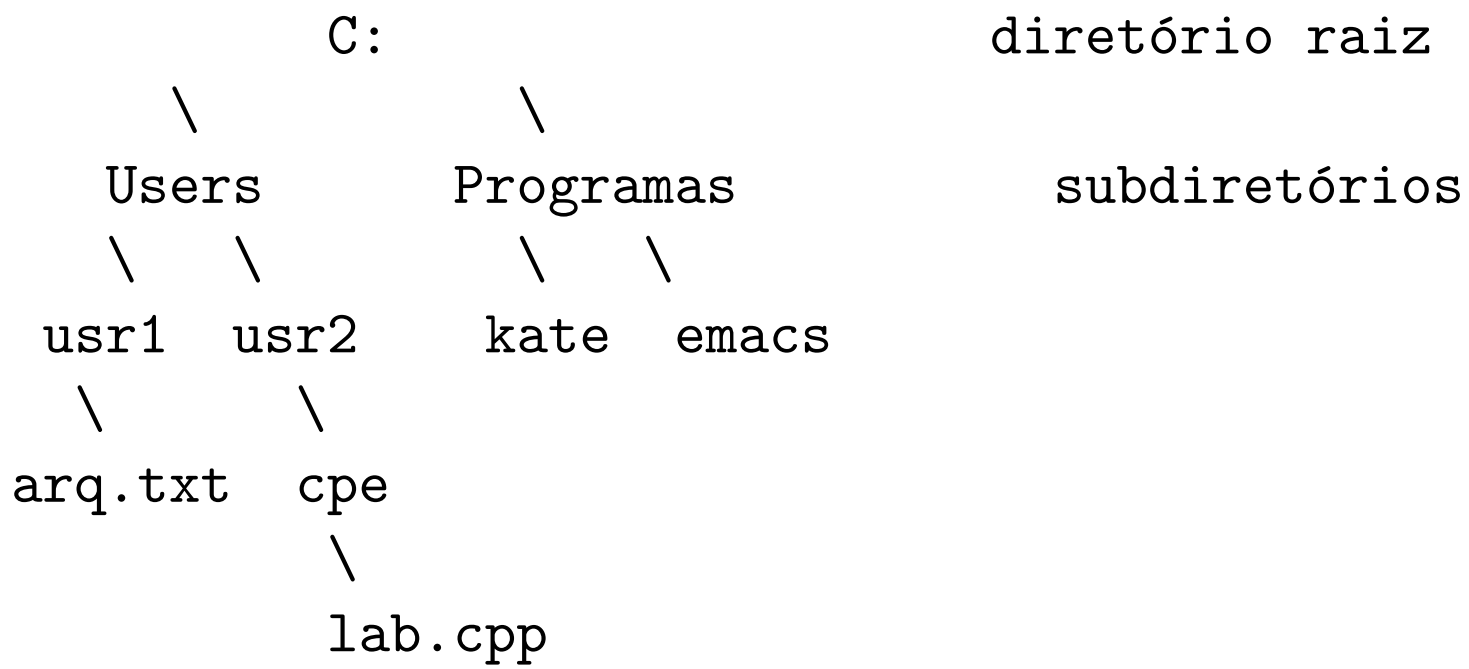
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte, documento texto simples, páginas HTML.

Arquivo binário: Seqüência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos em pdf, documentos do Word.

Diretório

- Também chamado de pasta.
- Contém arquivos e/ou outros diretórios.

Uma hierarquia de diretórios



Caminhos absolutos ou relativos

O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz. Os caminhos podem ser especificados de duas formas:

Caminho absoluto: descrição de um caminho desde o diretório raiz.

```
C:\Programas\emacs
C:\Users\usr1\arq.txt
```

Caminho relativo: descrição de um caminho desde o diretório corrente.

```
arq.txt
cpe\lab.cpp
```

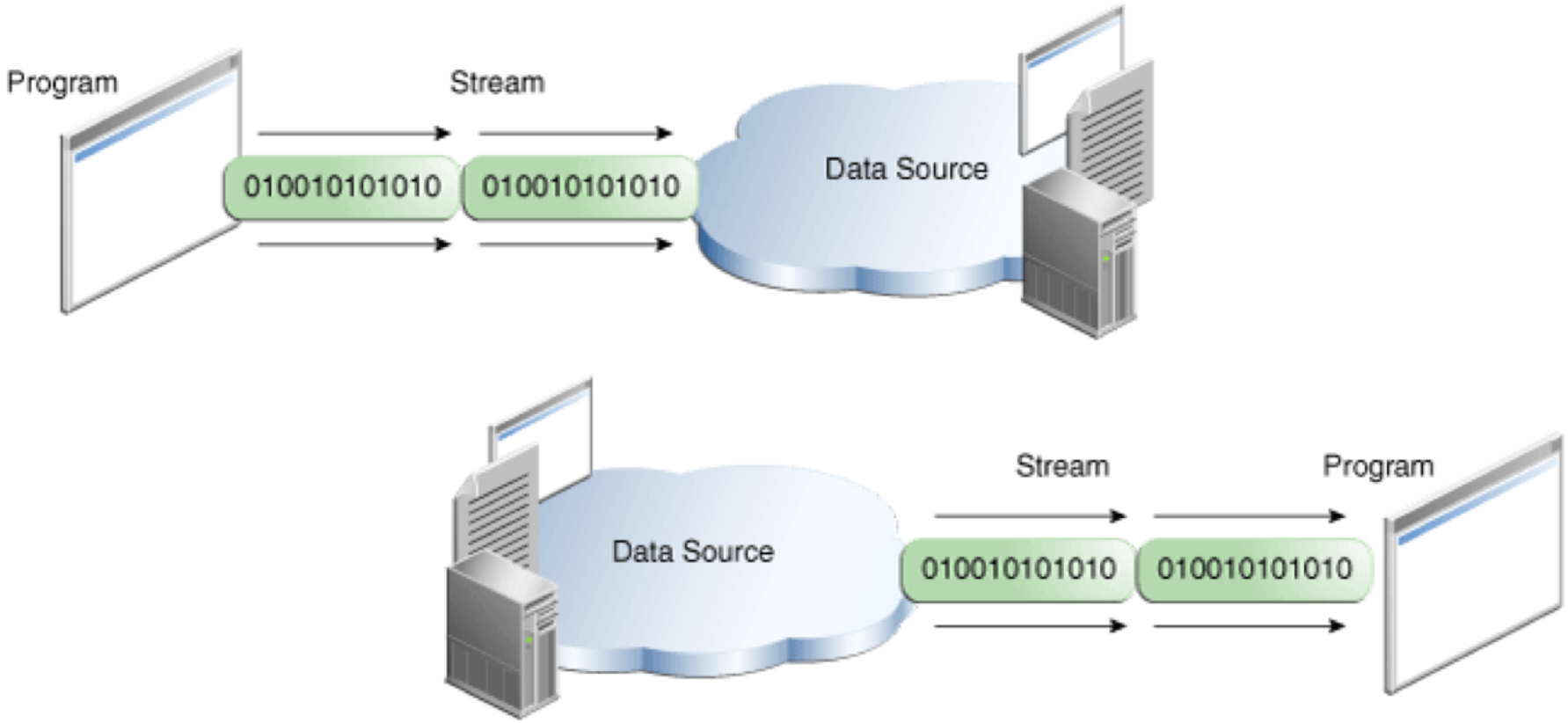
Atributos de arquivos

Além do nome, arquivos possuem vários outros atributos:

- Nome do arquivo
- Proprietário do arquivo
- Datas de criação, alteração e acesso
- Tamanho em bytes
- Permissão de acesso

Relembrando: Streams em C++

- Streams ou Fluxos são uma abstração que a linguagem C^{++} oferece para realizarmos operações de entrada (leitura de dados) e saída (escrita de dados)



Streams para arquivos

`ofstream`: classe de objetos stream para escrita em arquivos.
`ifstream`: classe de objetos stream para leitura de arquivos.
`fstream`: classe de objetos stream para escrita e leitura de arquivos.

Exemplo

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main () {
6    ofstream myfile;
7    myfile.open ("exemplo.txt");
8    myfile << "Escrevendo isto no arquivo.\n";
9    myfile.close();
10   return 0;
11 }
```

Abrindo um arquivo

- Antes de acessar um arquivo, devemos declarar uma variável do tipo `ifstream` (só leitura), `ofstream` (só escrita) ou do tipo `fstream` (ambos).

```
ifstream meuarquivo_paraler;
ofstream meuarquivo_paraescrever;
fstream  meuarquivo;
```

- Feito isto, chamamos a função-membro `open` associada à variável:

```
meuarquivo.open(<nome_arquivo>, <modo>);
```

Abrindo um arquivo para leitura

- <modo> é um parâmetro opcional com uma combinação das seguintes flags:

modo	operações
ios::in	leitura (padrão do tipo `ifstream`)
ios::out	escrita (padrão do tipo `ofstream`)
ios::binary	modo binário
ios::ate	posição inicial no fim do arquivo
ios::app	operações de escrita são concatenadas no fim do arquivo
ios::trunc	limpa arquivo antes de nova escrita

As flags podem ser combinadas com o operador OU bit-a-bit | . Exemplo:

```
fstream meuarquivo;
meuarquivo.open("exemplo.txt", ios::out | ios::app);
```

Lendo dados de um arquivo texto

- Após invocar a função `open`, podemos testar se a abertura foi bem sucedida chamando a função membro `is_open()`, que retorna `true` em caso positivo.
- Para ler dados de um arquivo-texto, usamos as mesmas operações usadas com o stream `cin`
- Para fechar o arquivo usamos a função-membro `close()`. É essencial fechar o arquivo depois de utilizá-lo.

Lendo dados do arquivo `teste.txt`

```
ifstream file;
file.open("teste.txt");
while (getline(file, s))
  cout << s << '\n';
file.close();
```

Veja o exemplo em `leia.cpp`.

Lendo dados de um arquivo texto

Note que a função função getline agora recebe como primeiro argumento na chamada o stream do arquivo a ser lido:

```
while (getline(file, s))
```

- O valor retornado pela função é uma referência ao próprio objeto stream, que caso seja avaliado como uma expressão booleana, é `true` se o stream está pronto para mais operações e `false` se o fim do arquivo foi alcançado ou algum outro erro ocorreu.

Escrevendo dados em um arquivo texto

- Para escrever em um arquivo texto, ele deve ser aberto de forma apropriada.
- Usamos as mesmas operações usadas com o stream `cout`.

Veja o exemplo em `escreva.cpp`.

Lendo caracteres numéricos de um arquivo texto como inteiros

```
fr.open("v-in.txt");
fr >> n;
v = new int[n];
for (i = 0; i < n; i++)
  fr >> v[i];
fr.close();
```

Veja o exemplo completo em `le_inteiros.cpp`.

Escrevendo inteiros em um arquivo texto

```
fw.open("v-out.txt", ios::trunc);
fw << n << '\n';
for (i = 0; i < n; i++)
  fw << v[i] << '\n';
fw.close();
```

Veja o exemplo em `le_inteiros.cpp`.

Escrevendo uma tabela de inteiros em um arquivo

- Como visto em tópico passado, a forma "tradicional" de fazer alocação dinâmica para matrizes consiste em criar `nlin` vetores de `ncol` inteiros.

```
v = new int*[nlin];
for (i = 0; i < nlin; i++)
  v[i] = new int[ncol];
for (i = 0; i < nlin; i++)
  for (j = 0; j < ncol; j++)
    fr >> v[i][j];
```

Veja o exemplo completo em `letabela.cpp`.

Parâmetros da função main()

- Vimos que o main() é a função principal de um programa, por onde ele sempre inicia sua execução.
- É possível passar argumentos para essa função também?
- SIM: a passagem de argumentos à função main() pode ser feita na linha de comando através das variáveis argc e argv.

Parâmetros da função main()

int main(int argc, char *argv[])

- A variável argc (argument count) contém o número de parâmetros passados através da linha de comando, incluindo o nome do programa.
- A variável argv (argument values) é um apontador para cadeias de caracteres (C-strings). Cada string é um argumento da linha de comando.

Argumentos para a função main()

int main(int argc, char *argv[])

Na linha de comando - exemplo:
./programa op1 op2 (Unix, Mac) ou
programa.exe op1 op2 (Windows)

Dentro do main:
argc = 3
argv[0] = string "./programa"ou "programa.exe"
argv[1] = string "op1"
argv[2] = string "op2"

Exemplo - Argumentos para o main

- Um programa que copia arquivos e recebe os nomes através da linha de comando.

```
fr.open(argv[1], ios::in);
if (!fr.is_open()) {
  cerr << "ERRO ao tentar abrir: " << argv[1];
  return 2;
}
fw.open(argv[2], ios::out | ios::app);
```

Veja o exemplo completo em `copiar.cpp`.